Analisa Kompatibilitas Dan *Quality Of Service* Komunikasi Audio Call Pada WebRTC dengan Menggunakan *Framework* EasyRTC

M. Vicky Agassi[(1)], Rakhmadhany Primananda, S.T, M.Kom[(2)], Barlian Henryranu Prasetio, S.T, M.T[(3)]

Teknik Informatika, Fakultas Ilmu Komputer Universitas Brawijaya, Malang, Indonesia.

Email: mohammadvickyagassi[at]rocketmail.com[(1)], rakhmadhany[at]ub.ac.id[(2)], barlian[at]ub.ac.id[(3)]

Abstrak -- WebRTC adalah sebuah teknologi aplikasi berbasis web yang bersifat *opensource*, memungkinkan pengguna untuk berkomunikasi secara *real-time* tanpa perlu menginstall *plugin*, hanya menggunakan *browser* yang support untuk webRTC. Pada penelitian ini webRTC yang dibangun menggunakan framework easyRTC dan node.js sebagai media signalingnya, codec audio yang digunakan oleh framework easyRTC adalah codec opus. Tujuan dari penelitian ini adalah untuk mengetahui kompatibilitas dari webRTC audio call dari framework easyRTC dapat berjalan pada browser Chrome v52.0, Mozilla v38.0.5, Opera v38.0, Safari v5.1.7 dan Internet Explorer v8.0, dan untuk mengetahui *quality of service* (QoS) dari webRTC audio call, apakah nilai QoS yang dihasilkan dapat memenuhi standarisasi komunikasi yang baik dengan menurut TIPHON. Parameter *quality of service* yang digunakan dalam penelitian ini adalah rata-rata *delay*, *jitter*, *throughput* dan *packets loss*. Hasil yang diperoleh dari penelitian ini menujukkan bahwa hanya browser Mozilla v38.0.5. quality of service yang dihasilkan mempunyai rata-rata *delay* dengan nilai 9.977 ms menunjukkan indeks sangat baik dan nilai *jitter* 0.000314 ms yang dihasilkan juga masuk dalam indeks baik menurut versi TIPHON, namun masih mempunyai presentase *packets loss* 4.255% yang menunjukkan kategori indeks sedang dan nilai *throughput* 14.019 kbps yang dihasilkan masuk dalam indeks jelek menurut versi TIPHON. Sehingga tergolong cukup untuk kategori komunikasi menurut versi TIPHON.

Kata kunci : webRTC, easyRTC, node.js, *quality of service* (QoS).

1. PENDAHULUAN

Kemajuan teknologi dari masa ke masa telah berkembang begitu cepat, terutama dalam bidang komunikasi. Teknologi komunikasi sangat diperlukan dalam kehidupan dikarenakan dapat mempermudah komunikasi antar manusia walaupun dengan perbedaan jarak dan waktu. Saat ini telah banyak dikembangkan teknologi komunikasi yang menggunakan IP dengan kata lain *Voice Over Internet Protocol* atau biasa disebut dengan VoIP (Muhammad Iqbal dkk, 2012). Sebelumnya, kebanyakan untuk mendapatkan komunikasi VoIP pengguna harus menginstall *plugin* terlebih dahulu agar dapat menggunakan layanan VoIP tersebut. Contoh layanan VoIP yang harus menginstall *plugin* adalah Skype, Facebook Messenger, Line dan sejenisnya. Namun sekarang telah dikembangkan suatu teknologi yang dapat menggunakan layanan VoIP tanpa harus menginstall *plugin*, teknologi tersebut adalah WebRTC (*Web Real-Time Communication*) (Edholm Phil etal, 2014).

WebRTC merupakan sebuah teknologi aplikasi berbasis web yang bersifat *opensource*, memungkinkan pengguna untuk berkomunikasi secara *real-time* tanpa perlu menginstall *plugin*, hanya menggunkan *browser* yang support untuk webRTC, sehingga pengguna dapat berkomunikasi dengan pihak lain. WebRTC memanfaatkan fitur dari HTML5 dan javascript sehingga memungkinkan untuk mengaktifkan audio yang tertanam pada *browser* dan dapat berkomunikasi secara visual dalam browser (Phil Edholm E. at.). Meskipun webrtc adalah *peer-to-peer* namun

dalam arsitektur webrtc memerlukan server sebagai media untuk *signaling* (Manson Rob, 2013). Pada penelitian webRTC ini menggunakan module Node.js yang didalamnya sudah terinstall socket.io sebagai media untuk *signaling* pada webrtc sehingga antar pengguna webrtc dapat terhubung dan dapat berkomunikasi (McLaughlin Brett, 2011).

Saat terjadinya proses komunikasi antar pengguna satu dengan pengguna lainnya, proses pengiriman data akan terus berjalan dan akan berhenti jika salah satu pengguna menghentikan komunikasi tersebut. Dengan begitu data akan semakin banyak atau semakin menumpuk, hal ini memungkinkan nilai jitter semakin meningkat sehingga komunikasi antar pengguna akan terganggu dan mengakibatkan pengguna tidak nyaman melakukan komunikasi pada webrtc (Dewi Yolanda S. A. dkk, 2013).

Dari permasalahan diatas penulis ingin melakukan analisa terhadap layanan VoIP webrtc dengan *framework* easyrtc yang menggunakan node.js sebagai media *signaling* serta untuk mengetahui *browser* apa saja yang *support* dengan *framework* easyrtc yang menggunakan node.js sebagai media *signaling*. Webrtc audio call diimplementasikan berjalan pada jaringan lokal yang mempunyai bandwidth 128 kbps. Untuk mengetahui baik atau buruk dari layanan VoIP menggunakan metode *Quality Of Service* atau biasa disebut dengan *QoS* pada layanan webrtc. Parameter QoS yang gunakan adalah rata-rata *delay, throughput, jitter, dan packets loss.* Parameter *QoS* tersebut mengacu pada versi TIPHON untuk mengetahui apakah layanan webrtc yang diteliti telah memenuhi syarat *QoS* yang baik.

2. KAJIAN PUSTAKA

2.1 API Javascript WebRTC

Terdapat beberapa rumus untuk menghitung beberapa parameter dari QoS yaitu *Bandwidth, Delay, Jitter, Throughput, Packets Loss*.

- *Bandwidth*

Bandwidth adalah sebagai tolak ukur kecepatan dari banyaknya informasi yang dapat mengalir pada *source* ke *destination*. Jenis *bandwidth* ini biasanya diukur dalam bentuk kbps (kilo bit per second). Rumus untuk menghitung bandwidth dapat dilihat pada persamaan 1 dibawah ini.

$$Bandwidth = \frac{\text{ukuran data}}{\text{waktu unduh terbaik}}$$

Persamaan 1 *Bandwidth*

- Ukuran data : total data yang berjalan pada suatu jaringan, dengan satuan bit.
- Waktu unduh terbaik : total waktu saat mengunduh suatu data dengan satuan *second* (Yanto. 2014).

- *Delay*

Delay merupakan waktu yang dibutuhkan suatu paket data untuk sampai dari *source* ke *destination*. Faktor yang dapat mempengarui nilai dari *delay* adalah jarak, media fisik, kongesti, atau proses rentan waktu yang lama. Rumus untuk menghitung rata-rata *delay* dapat dilihat pada persamaan 2 dibawah ini.

$$Rata - rata\ Delay = \frac{\text{Total Delay}}{\text{Total paket yang diterima}}$$

Persamaan 2 *Delay*

- Total Delay : Total delay pada saat melakukan transfer paket data dari *source* ke *destination*.
- Total paket yang diterima : jumlah paket data yang dapat diterima pada waktu tertentu (Yanto. 2014).

- Jitter

Jitter merupakan variasi delay dari kedatangan paket data ke destinasi. Faktor yang dapat mempengaruhi jitter adalah perubahan pada peningkatan traffic pada jaringan sehingga menyebabkan penyempitan pada bandwidth dan menyebabkan antrian paket data. Rumus untuk menghitung jitter dapat dilihat pada persamaan 3 dibawah ini.

$$\text{Jitter} = \frac{\text{Total variasi delay}}{\text{Paket data yang diterima} - 1}$$

Persamaan 3 *Jitter*

- Total variasi delay : total dari variasi delay yang didapat. Untuk mencari variasi delay menggunakan rumus seperti pada persamaan 4 dibawah ini.

$$Variasi\ Delay = (delay\ 2 - delay\ 1) + (delay\ 3 - delay\ 2) + \ldots + (delay\ n - delay\ (n-1))$$

Persamaan 4 Variasi *Delay*

- Paket data yang diterima : jumlah paket data yang dapat diterima pada waktu tertentu (Yanto. 2014).

- Throughput

Throughput merupakan bandwidth yang diukur dengan satuan waktu tertentu pada saat melakukan transfer data. Rumus yang digunakan untuk menghitung throughput dapat dilihat pada persamaan 5 dibawah ini.

$$Throughput = \frac{\text{Paket data yang diterima}}{\text{Lama pengamatan}}$$

Persamaan 5 *Throughput*

- Paket data yang diterima : jumlah paket data yang diterima dalam kurun waktu tertentu dengan satuan bytes.
- Lama pengamatan : interval waktu yang dilakukan untuk mengamati proses pengiriman paket data (Yanto. 2014).

- Packets Loss

Packets Loss merupakan sebuah kondisi atau parameter yang menunjukkan adanya jumlah paket data yang hilang saat dikirim dari source ke destination. Rumus yang digunakan untuk menghitung *packets loss* dapat dilihat pada persamaan 6 dibawah ini.

$$\text{Packets Loss} = \frac{(\text{Tot pkt data dikirim} - \text{Tot pkt data diterima})}{\text{Total paket data yang dikirim}} \text{ x } 100\%$$

Persamaan 6 *Packets Loss*

- Total paket data yang dikirim : jumlah paket data yang dapat dikirim pada waktu tertentu.
- Total paket data yang diterima : jumlah paket data yang dapat diterima pada waktu tertentu (Muhammad Yanto. 2014).

3. METODOLOGI

Melakukan uji kompatibilitas sistem pada lima browser, yaitu chrome v52.0, Mozilla v38.0.5, opera v38.0, safari v5.1.7, dan internet explorer v8.0. serta melakukan metode *quality of service* yang mengacu pada standarisasi versi TIPHON, sehingga dapat mengetahui hasil dari monitoring rata-rata delay, jitter, throughput, dan packets loss yang nantinya dapat dijadikan tolak ukur dari kinerja webrtc apakah tergolong baik atau jelek. Empat parameter tersebut yang biasa digunakan untuk mengukur *quality of service* yang berbasis IP yang ada pada suatu jaringan (William S. Bubanto dkk, 2014). Berikut adalah standarisasi dari versi TIPHON.

Nilai	Persentase (%)	Indeks
3.8 – 4	95 – 100	Sangat Baik
3 – 3.79	75 – 94.75	Baik
2 – 2.99	50 – 74.75	Kurang Baik
1 – 1.99	25 – 49.75	Jelek

Tabel Error! No text of specified style in document.**.1** Indeks Parameter QoS (Sumber TIPHON)

Kategori Delay	Besar Delay	Indeks
Sangat Baik	< 150 ms	4
Baik	150 – 300 ms	3
Sedang	300 – 450 ms	2
Jelek	> 450 ms	1

Tabel Error! No text of specified style in document.**.2** Standarisasi Delay (Sumber TIPHON)

Kategori Degradasi	Packet loss	Indeks
Sangat Baik	0%	4
Baik	3%	3
Sedang	15%	2
Jelek	25%	1

Tabel Error! No text of specified style in document.**.3** Standarisasi Packets Loss (Sumber TIPHON)

Kategori Throughput	Throughput	Indeks
Sangat Baik	100%	4
Baik	75%	3
Sedang	50%	2
Jelek	< 25%	1

Tabel Error! No text of specified style in document.**.4** Standarisasi Throughput (Sumber TIPHON)

Kategori Degradasi	Peak Jitter	Indeks
Sangat Baik	0 ms	4
Baik	0 – 75 ms	3
Sedang	76 – 125 ms	2
Jelek	125 – 225 ms	1

Tabel Error! No text of specified style in document.**.5** Standarisasi Jitter (Sumber TIPHON)

Tabel 3.1, 3.2, 3.3, 3.4 dan 3.5 digunakan sebagai acuan standarisasi komunikasi yang baik menurut versi TIPHON. Untuk menghitung QoS dengan parameter *delay*, jitter, throughput, dan *packets loss* menggunakan rumus sebagai berikut :

- Delay

Delay merupakan waktu yang dibutuhkan suatu paket data untuk sampai dari source ke destination. Factor yang dapat mempengarui nilai dari delay adalah jarak, media fisik, kongesti, atau proses rentan waktu yang lama. Rumus untuk menghitung rata-rata delay seperti pada Persamaan 2 *Delay*.

- Jitter

Jitter merupakan variasi delay dari kedatangan paket data ke destinasi. Faktor yang dapat mempengaruhi jitter adalah perubahan pada peningkatan traffic pada jaringan sehingga menyebabkan penyempitan pada bandwidth dan menyebabkan antrian paket data. Rumus untuk menghitung jitter seperti pada Persamaan 3 *Jitter* dan untuk menghitung total variasi delay menggunakan Persamaan 4 Variasi *Delay*.

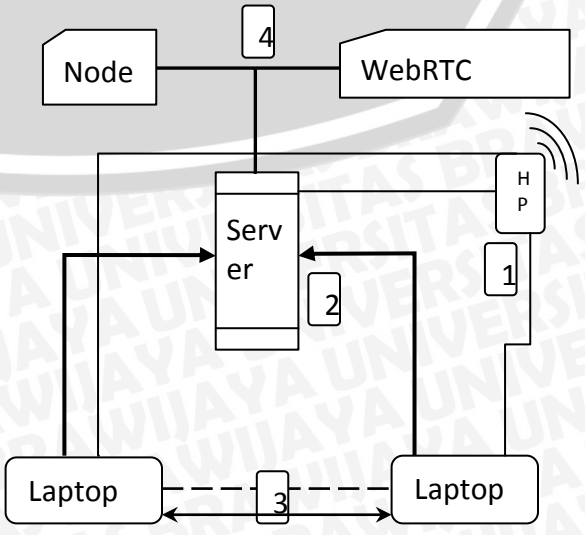
- Throughput

Throughput merupakan bandwidth yang diukur dengan satuan waktu tertentu pada saat melakukan transfer data. Rumus yang digunakan untuk menghitung throughput seperti pada Persamaan 5 *Throughput*.

- Packets Loss

Packets Loss merupakan sebuah kondisi atau parameter yang menunjukkan adanya jumlah paket data yang hilang saat dikirim dari source ke destination. Rumus yang digunakan untuk menghitung *packets loss* seperti pada Persamaan 6 *Packets Loss*. Yang nantinya akan penulis sesuaikan dengan standarisasi dari versi TIPHON, sehingga dapat mengetahui *Quality of Service* dari audio call webrtc ini.

4. PERANCANGAN SISTEM



Gambar 4.1 Rancangan Arsitektur WebRTC

Pada rancangan arsitektur gambar 4.1 terdapat tiga komponen perangkat dalam arsitektur tersebut, yaitu :

1. Device Handphone

Device handphone (HP) digunakan untuk tethering, sehingga *device* yang *connect* ke *hotspot* HP tersebut akan mendapatkan *IP address*, tentunya setiap *device* akan mendapatkan *IP address* yang berbeda.

2. Laptop A

Pada laptop A menggunakan *Operating System* (OS) windows 7. IP address yang didapatkan laptop A 192.168.43.203.

3. Laptop B

Pada laptop B hanya terdapat satu OS, yaitu OS windows. Laptop B mendapatkan IP address 192.168.43.65.

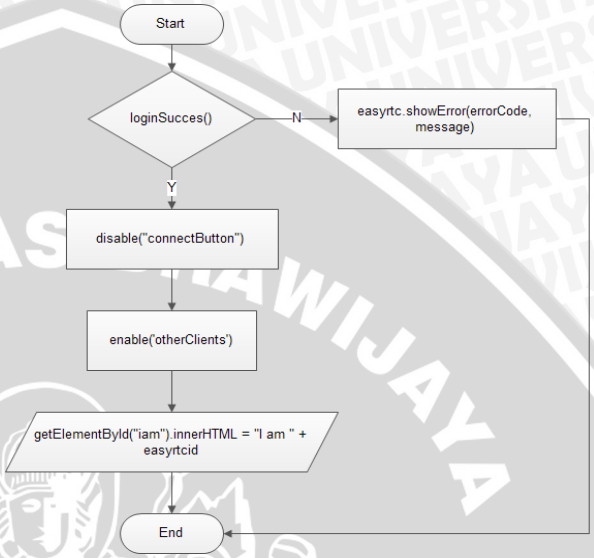
4. Server

Pada Server menggunakan OS Ubuntu 14.04. Pada server ini didalamnya terdapat modul node.js, socket.io dan *framework* easyRTC.

Node.js dan socket.io tersebut digunakan untuk proses *signaling* dan proses untuk mengakses webRTC *audio call* pada port 8181. IP address yang digunakan server 192.168.43.215.

Penjelasan tentang empat komponen tersebut berkaitan dengan empat point yang terdapat pada gambar 4.1, berikut adalah flowchart dari proses login pada webrtc, proses terbentuknya komunikasi dan proses hangup.

- Flowchart Proses Login

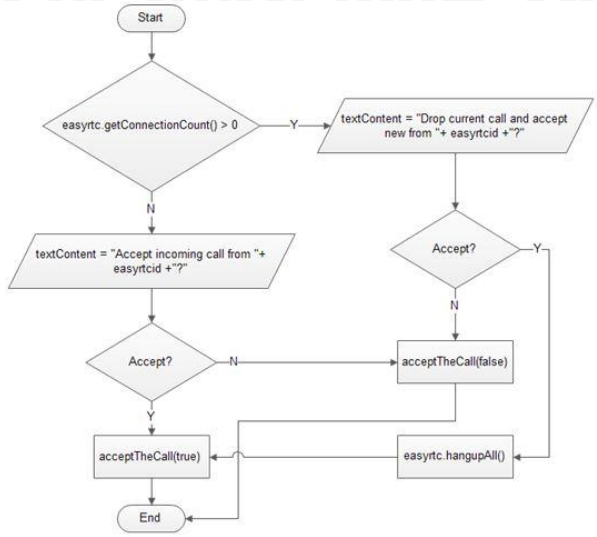


Gambar Error! No text of specified style in document..**1** Flowchart Proses Login

mana
telah

dapat mengakses webRTC audio call pada server. Pada tahap ini laptop A dan laptop B harus melakukan registrasi terlebih dahulu dengan memilih tombol *connect* pada webRTC, saat melakukan penggunaka menekan tombol *connect*, method yang dijalankan pada sistem webrtc adalah loginSuccess(). Node.js akan meminta izin untuk mengakses media mikrofon pada laptop A dan laptop B, apabila browser dari pengguna tidak *support* dengan HTML5 dan javascript dari webRTC maka sistem akan menampilkan pesan error, jika browser dari pengguna *support* dengan HTML5 dan javascript dari webRTC maka tombol *connect* akan menjadi *disable* dan sistem akan menampilkan pengguna lain yang telah login pada sistem serta sistem memberikan sebuah ID pada pengguna, dimana ID tersebut berguna sebagai variable untuk melakukan komunikasi terhadap pengguna lain, karena setiap pengguna yang login pada sistem akan mendapatkan ID yang berbeda.

- Flowchart Proses Terjadinya Komunikasi

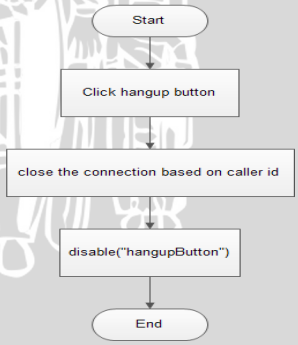


Gambar Error! No text of specified style in document.**.2** Flowchart Proses Terjadinya Komunikasi

pengguna laptop A ingin menelepon pengguna laptop B, maka laptop A mengirimkan informasi berupa ID nya. Informasi tersebut digunakan untuk proses signaling yang dilakukan oleh node.js untuk mencari node yang dituju oleh pengguna laptop A. Pada tahap ini sistem akan mendeteksi banyaknya koneksi yang sedang ingin menelepon pengguna laptop B. Apabila method easyrtc.getConnetionCount() yang didapat lebih dari 0, maka pada pengguna laptop B mendapat notifikasi dari sistem, bahwa ada yang ingin menelpon dengan ID A, apakah anda ingin menerima?. Jika pengguna laptop B menerima panggilan dari pengguna laptop A maka proses komunikasi sebelumnya akan di hentikan dengan method hangupAll() dan berganti dengan komunikasi dengan pengguna laptop A, namun jika pengguna laptop B tidak menerima atau pengguna laptop B me-reject panggilan dari pengguna laptop A, maka proses panggilan dari pengguna laptop A ke pengguna laptop B akan dihentikan oleh method acceptTheCall(false).

Jika getConnectionCount kurang dari 0 yang artinya sedang tidak ada aktifitas komunikasi pada pengguna laptop B, maka pada pengguna laptop B mendapatkan notifikasi dari sistem bahwa ada yang meneleponnya dengan ID pengguna A, serta terdapat dua tombol yaitu *accept* dan *reject*. Apabila pengguna laptop B membatalkan panggilan dari pengguna laptop A, maka sistem menjalankan method AcceptTheCall(false), sehingga proses sambungan telepon dari pengguna A ke pengguna B akan terputus. Jika pengguna laptop B menerima panggilan dari pengguna laptop A, maka sistem menjalankan method accpetTheCall(true), sehingga terbentuklah ACK dari kedua pengguna tersebut dan dapat melakukan komunikasi.

- Flowchart Proses Hangup



Gambar Error! No text of specified style in document.**.3** Flowchart Proses Hangup

at salah
engguna
laptop B ingin mengakhiri panggilan teleponnya, dimana mereka telah melakukan proses komunikasi yang telah dijelaskan pada gambar 4.2. Pengguna tersebut dapat menekan tombol *hangup*, lalu sistem akan menutup koneksi dari

caller ID yang bersangkutan dan tombol *hangup* akan menjadi *disable*.

5. IMPLEMENTASI

Diperlukannya modul node.js sebagai proses *signaling*, pada penelitian ini dibutuhkan node.js pada server dimana server tersebut berada dalam OS ubuntu 14.04. langkah-langkah untuk menginstall node.js pada server dapat dilihat pada Source Code 1.

```
#curl                                   -sL
https://deb.nodesource.com/setup
| sudo bash -
#apt-get install -y nodejs
https://github.com/priologic/eas
yrtc.git
```

Source Code 1 Install Node.js

Setelah node.js selesai terinstal langkah selanjutnya adalah instalasi *framework* easyrtc pada server dengan meletakkan *framework* easrtc pada direktori var/nodes/easyrtc.

```
$sudo su
#svn checkout
https://github.com/priologic/ea
syrtc.git
#cd easyrtc.git/trunk

#mkdir /var/nodes
#mkdir /var/nodes/easyrtc
#cp -a server_example
#cd
/var/nodes/easyrtc/server_examp
le
#npm install
#cd
node_modules/easyrtc/server_exa
mple
#npm install
```

Source Code 2 Install Framework EasyRTC

Setelah *framework* easyrtc selesai terinstal, maka selanjutnya memanfaatkan *javascript* dan css pada *framework* easyrtc, dimana folder tersebut dipindahkan ke direktori /var/nodes/vicky/easyrtc/server_example/node_modules/easyrtc/server_example/static. Langkah-langkahnya untuk memindahkan direktori yang dimaksud dapat dilihat pada Source Code 3.

```
#cd
/var/nodes/vicky/easyrtc/server_
example/node_modules/easyrtc/dem
os
#cp -a css
/var/nodes/easyrtc/server_exampl
e/node_modules/easyrtc/server_ex
ample/static
#cp -a js
/var/nodes/easyrtc/server_exampl
e/node_modules/easyrtc/server_ex
ample/static
# cp -a images
/var/nodes/easyrtc/server_exampl
e/node_modules/easyrtc/server_ex
ample/static
```

Source Code 3 Konfigurasi Folder Pada EasyRTC

Agar pengguna dapat mengakses webrtc *audio call* maka diperlukannya port untuk alamat webRTC *audio call*. Pada webRTC ini memakai port 8181. Berikut adalah konfigurasi pada file server.js pada direktori /var/nodes/easyrtc/server_example/node_modules/easyrtc/server_example.

```
// Memanggil modul yang
dibutuhkan
var http    = require("http");
// http server core module
var express =
require("express");  // web
framework external module
var io      =
require("socket.io");  // web
socket external module
var easyrtc = require("../");
// letak direktori modul
eksternal

// mensetting dan konfigurasi
untuk merujuk ke folder static
pada saat mengakses web.
var httpApp = express();
httpApp.use(express.static(__di
rname + "/static/"));

// mensetting Express http
```

```
server pada port 8080
var webServer =
http.createServer(httpApp).listen(8181);

});
```

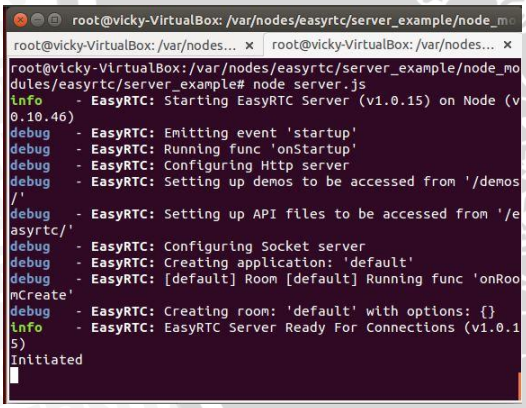
Source Code 4 Konfigurasi File server.js

Setelah semua selesai diseting dan dikonfigurasi, sekarang tahap untuk menghidupkan *signaling* dari server, untuk menjalankan *signaling* agar webrtc tersebut dapat diakses oleh pengguna sebagai berikut

```
#cd /var/nodes/vicky/easyrtc
#node server.js
```

Source Code 4 Sintak Menjalankan node.js

Maka proses signaling akan berjalan dan server menunggu pengguna yang akan mengakses webrtc *audio call* pada IP address server dengan port 8181, tampilan dari proses *signaling* node.js dapat dilihat pada gambar 5.1



Gambar Error! No text of specified style in document.**.4** Proses *Signaling*

6. PE

6.1 Pengujian Kompatibilitas Sistem

Pada pengujian kompatibilitas webrtc *audio call* dengan *framework* easyrtc yang telah dilakukan pada kelima *browser* tersebut, dapat menganalisa bahwa *framework* easyrtc hanya dapat berjalan dengan lancar pada *browser* Mozilla, sedangkan pada *browser* lain mempunyai banyak kendala seperti javascript API getUserMedia() pada easyrtc mengalami *error* saat dijalankan pada *browser* chrome dan opera, lalu pada *browser* safari tidak *support* dengan webrtc dikarenkan HTML5 pada *browser* safari masih belum *support* webrtc, namun masih dapat mengakses webrtc *audio call* pada *browser*, hal ini menunjukkan bahwa safari masih belum mampu menjalankan *source code* HTML5 pada *framework* easyrtc namun masih dapat menjalankan *source code* javascript pada *framework* easyrtc, dan pada *browser* internet explorer tidak dapat mengakses webrtc *audio call* dikarenakan HTML5 dan javascript pada *browser* tersebut tidak dapat menjalankan *source code* javascript dan HTML5 pada *framework* easyrtc. Berikut hasil pengujian kompatibilitas yang telah dilakukan dapat dilihat pada tabel 6.1 dibawah ini.

Browser	**HTML5 audio**	**Websocket**	**WebRTC**
Chrome v52.0	✓	✓	X
Mozilla v38.0.5	✓	✓	✓
Opera v38.0	✓	✓	X
Safari v5.1.7	X	✓	X
Internet Explorer v8.0	X	X	X

Tabel **Error! No text of specified style in document.**.1 **Hasil Analisa Kompatibilitas**

6.2 Pengujian Quality of Service

Disinilah inti dari penelitian ini, menganalisa kompatibilitas dan *quality of service* webrtc *audio call* menggunakan *framework* easyrtc dengan *codec* opus dan node.js sebagai media signalingnya. Dari beberapa pengujian parameter

QoS *delay*, *jitter*, *throughput* dan *packets loss* selama 2 sampai 8 menit yang telah dilakukan, dari *sample* yang didapat pada setiap rentan waktunya, yaitu 2 menit, 4 menit, 6 menit dan 8 menit. Dari nilai *sample* tersebut dapat digunakan untuk menghitung nilai *quality of service* dari rentan waktu 2 sampai 8 menit.

Pada tahap ini akan melakukan perhitungan untuk merata-rata hasil dari setiap parameter pada rentan waktu 2 sampai 8 menit. Hasilnya akan disesuaikan dengan standarisasi komunikasi yang baik versi TIPHON.

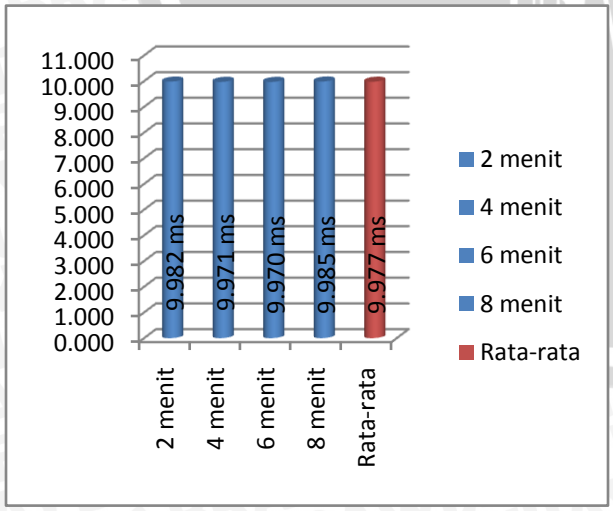
- **Menghitung Rata-rata *Delay* 2 s/d 8 menit**

Setelah mendapatkan sample rata-rata delay pada setiap rentan waktu. Diketahui bahwa nilai rata-rata delay pada setiap rentan waktu 2 s/d 8 menit antara 9.970 ms sampai 9.985 ms. Sehingga didapatkan hasil nilai dari rata-rata delay 2 s/d 8 menit dengan rumus

$$rata - rata = \frac{(2\text{ menit} + 4\text{ menit} + 6\text{ menit} + 8\text{menit})}{4}$$

Persamaan 7 Menghitung Rata-rata Nilai dari Keempat Parameter Qos

Dari perhitungan diatas yang menggunakan persamaan 7 menghasilkan nilai rata-rata delay 9.977 ms. Adapun grafik yang menunjukkan nilai rata-rata delay pada setiap rentan waktu 2 s/d 8 menit.

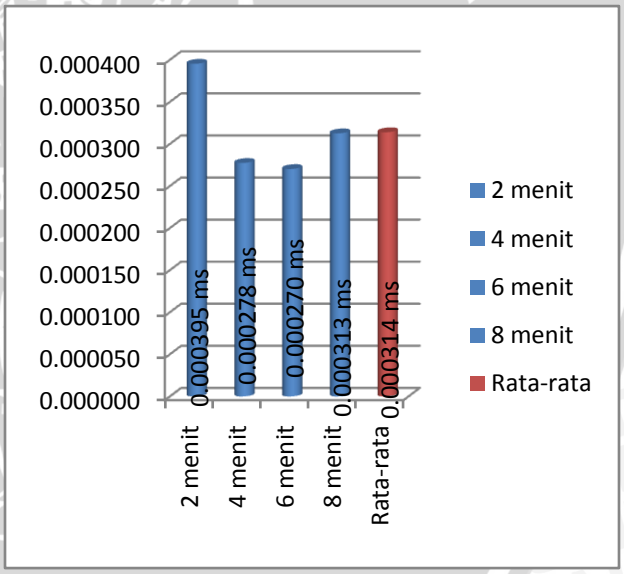


Gambar Error! No text of specified style in document.**.5** Grafik Nilai Rata-rata Delay Pada Rentan Waktu 2 s/d 8 menit

Pada gambar 6.2.1 grafik yang berwarna merah menunjukkan nilai rata-rata delay 9.977 ms, nilai rata-rata delay tersebut untuk dibandingkan dengan quality of service versi TIPHON.

- **Menghitung Nilai *Jitter* 2 s/d 8 menit**

Pada perhitungan nilai jitter yang telah dilakukan di setiap rentan waktunya. Diketahui bahwa nilai rata-rata jitter pada setiap rentan waktu 2 s/d 8 menit antara 0.000270 ms sampai 0.000395 ms. Sehingga didapatkan hasil nilai dari rata-rata jitter 2 s/d 8 menit dengan persamaan 7. sehingga menghasilkan nilai rata-rata jitter 0.000314 ms. Adapun grafik yang menunjukkan nilai rata-rata jitter pada setiap rentan waktu 2 s/d 8 menit.

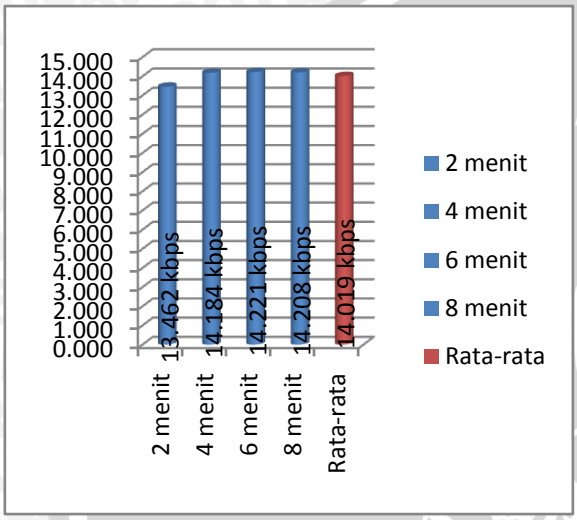


Gambar Error! No text of specified style in document.**.6** Grafik Nilai Rata-rata Jitter Pada Rentan Waktu 2 s/d 8 menit

Pada gambar 6.2.2 grafik yang berwarna merah menunjukkan nilai rata-rata jitter 0.000314 ms, nilai rata-rata jitter tersebut untuk dibandingkan dengan quality of service versi TIPHON.

- **Menghitung Nilai *Throughput* 2 s/d 8 menit**

Untuk perhitungan nilai throughput yang telah dilakukan di setiap rentan waktunya. Diketahui bahwa nilai rata-rata throughput pada setiap rentan waktu 2 s/d 8 menit antara 13.462 kbps sampai 14.221 kbps. Sehingga didapatkan hasil nilai dari rata-rata throughput 2 s/d 8 menit dengan persamaan 7. sehingga menghasilkan nilai rata-rata throughput 14.019 kbps. Adapun grafik yang menunjukkan nilai rata-rata throughput pada setiap rentan waktu 2 s/d 8 menit.

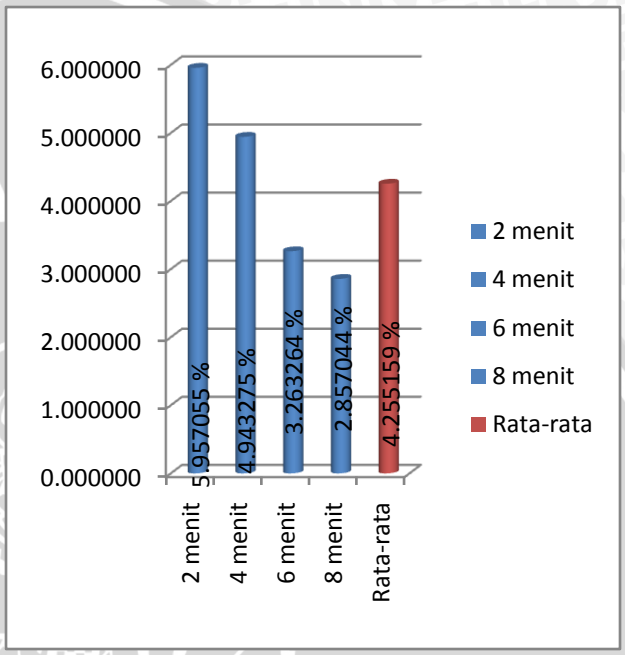


Gambar Error! No text of specified style in document..**7** Grafik Nilai Rata-rata Throughput Pada Rentan Waktu 2 s/d 8 menit

merah menunjukkan nilai rata-rata throughput 14.019 kbps, nilai rata-rata throughput tersebut untuk dibandingkan dengan quality of service versi TIPHON.

- **Menghitung Nilai *Packets Loss* 2 s/d 8 menit**

Pada perhitungan nilai packets loss yang telah dilakukan di setiap rentan waktunya. Diketahui bahwa nilai rata-rata packets loss pada setiap rentan waktu 2 s/d 8 menit antara 2.857044 %sampai 5.957055 %. Sehingga didapatkan hasil nilai dari rata-rata packets loss 2 s/d 8 menit dengan persamaan 7. Sehingga menghasilkan nilai rata-rata packets loss 4.255159 %. Adapun grafik yang menunjukkan nilai rata-rata packets loss pada setiap rentan waktu 2 s/d 8 menit.



Gambar Error! No text of specified style in document..**2.4** Grafik Nilai Rata-rata Packets Loss Pada Rentan Waktu 2 s/d 8 menit

merah menunjukkan nilai rata-rata packets loss 4.255159 %, nilai rata-rata packets loss tersebut untuk dibandingkan dengan quality of service versi TIPHON.

- **Hasil Uji Coba Dalam Rentan Waktu 2 s/d 8 menit**

Setelah mendapatkan nilai rata-rata dari setiap parameter quality of service, yaitu rata-rata delay, jitter, throughput dan packets loss, maka didapatkan hasil dari pengujian quality of service webrtc audio call dengan framework easyrtc yang menggunakan codec opus dengan node.js sebagai media signalingnya pada rentan waktu 2 s/d 8 menit. Hasil tersebut ditampilkan pada tabel 6.2.1 dibawah ini.

Rata-rata QoS dalam rentan waktu 2 sampai 8 menit	
Kategori	Nilai
Rata-rata Delay (ms)	9.977
Jitter (ms)	0.000314
Throughput (kbps)	14.019
Packets Loss (%)	4.255159

Tabel Error! No text of specified style in document.**.6** Nilai Parameter QoS Yang Dihasilkan Pada Rentan Waktu 2 s/d 8 menit

6.3 Analisis Hasil Pengujian Dari Rentan Waktu 2 s/d 8 menit

Pada tahap ini melakukan perbandingan nilai dari parameter rata-rata delay, jitter, throughput dan packets loss yang didapatkan dari rangkaian percobaan yang telah dilakukan dengan quality of service versi TIPHON. Indeks dari standarisasi versi TIPHON dapat dilihat pada tabel 3.1.

- **Perbandingan Nilai Rata-rata Delay dengan versi TIPHON**

Pada point ini membandingkan nilai rata-rata delay yang didapatkan dari rangkaian percobaan yang telah dilakukan dengan standarisasi nilai rata-rata delay versi TIPHON. Standarisasi *delay* versi TIPHON dapat dilihat pada tabel 3.2.

Nilai rata-rata delay yang dihasilkan adalah 9.977 ms. Jika dibandingkan dengan standarisasi Delay pada versi TIPHON masuk kedalam indeks sangat baik, dikarenakan nilai rata-rata delay dari penelitian ini < 150 ms. Dapat disimpulkan bahwa webrtc audio call dengan menggunakan framework easyrtc dengan codec opus dan node.js sebagai media signalingnya untuk pertukaran data sangat baik hanya dalam rata-rata delay 9.977 ms dari rentan waktu 2 s/d 8 menit.

- **Perbandingan Nilai Jitter dengan versi TIPHON**

Pada point ini membandingkan nilai jitter yang dihasilkan dari rangkaian percobaan yang telah dilakukan dengan standarisasi nilai jitter versi TIPHON. Standarisasi nilai *jitter* versi TIPHON dapat dilihat pada tabel 3.5.

Nilai jitter yang dihasilkan adalah 0.000314 ms. Jika dibandingkan dengan standarisasi jitter pada versi TIPHON masuk kedalam indeks baik, dikarenakan nilai rata-rata jitter dari penelitian ini adalah 0 - 75 ms. Dapat disimpulkan bahwa webrtc audio call dengan menggunakan framework easyrtc dengan codec opus dan node.js sebagai media signalingnya, mampu menangani beban paket data yang bertambah semakin banyak, hal tersebut dapat dilihat dari record data yang diterima pada rentan waktu 2 s/d 8 menit. Perlu diketahui faktor meningkatnya nilai jitter adalah semakin banyaknya beban paket data yang diterima pada jaringan (Dewi Yolanda S. A. dkk, 2013).

- **Perbandingan Nilai throughput dengan versi TIPHON**

Pada point ini membandingkan nilai throughput yang dihasilkan dari rangkaian percobaan yang telah dilakukan dengan standarisasi nilai rata-rata delay versi TIPHON. Standarisasi *throughput* versi TIPHON dapat dilihat pada tabel 3.4.

Nilai throughput yang dihasilkan adalah 14.019 kbps. Perlu diketahui pada penelitian ini hanya menggunakan bandwidth 128 kbps. Sehingga dari 128 kbps throughput yang dihasilkan adalah 14.128 kbps, maka diperlukannya persamaan untuk mengubah menjadi nilai persentase, dikarenakan standarisasi throughput versi TIPHON bernilai persentase. Persamaan untuk mengubah nilai *throughput* menjadi persentase dapat dilihat pada persamaan 8.

$$\text{Throughput}\,(\%) = \frac{\text{Throughput}}{Bandwidt\ h} x100$$

Persamaan 8 Persentase Throughput

$$(\%) = \frac{14.019}{128} x100$$

Hasil perhitungan throughput dari satuan kbps ke satuan persentase menghasilkan nilai 10.952%. hasil tersebut menunjukkan bahwa dari bandwidth 128 kbps hanya menghasilkan throughput 14.128 kbps. Sehingga jika persentasekan mendapatkan nilai throughput 10.952%.

Jika dibandingkan dengan standarisasi throughput pada versi TIPHON masuk kedalam indeks jelek, dikarenakan nilai throughput dari penelitian ini <25%. Dapat disimpulkan bahwa dengan menggunakan bandwidth 128 kbps webrtc audio call dengan menggunakan framework easyrtc dengan codec opus dan node.js sebagai media signalingnya, dalam melakukan pengiriman data yang sukses sampai ketujuan masih jelek.

- **Perbandingan Nilai Packets Loss dengan versi TIPHON**

Pada point ini membandingkan nilai throughput yang dihasilkan dari rangkaian percobaan yang telah dilakukan dengan standarisasi nilai rata-rata delay versi TIPHON. Standarisasi *packets loss* versi TIPHON dapat dilihat pada tabel 3.3.

Nilai packets loss yang dihasilkan adalah 4.255159%. Jika dibandingkan dengan standarisasi packets loss pada versi TIPHON masuk kedalam indeks sedang, dikarenakan nilai packets loss dari penelitian ini > 3%. Hal ini berbanding lurus dengan nilai throughput yang dihasilkan, nilai throughput yang dihasilkan pada penelitian ini masuk kedalam indeks jelek pada standarisasi versi TIPHON.

7. KESIMPULAN

Berdasarkan penelitian webRTC yang telah dilakukan yaitu menganalisa framework easyRTC dan node.js sebagai media signaling serta codec opus yang digunakan untuk mentransmisi suara, dapat disimpulkan bahwa :

1. Dari lima browser yang telah dicoba untuk menguji kompatibilitas webRTC yaitu browser chrome v52.0, Mozilla firefox v38.0.5, opera v38.0, safari v5.1.7 dan internet explorer v8.0, webRTC dapat berjalan dengan lancar hanya pada browser Mozilla firefox. Selain browser Mozilla firefox terdapat error, error yang ditampilkan oleh sistem adalah sebagai berikut :

- Browser Chrome v52.0

"*Failed to get access to local media. Error code was PermissionDeniedError*". Hal ini menunjukkan bahwa saat sistem meminta izin untuk mengakses media mikrofon javascript dari *framework* easyrtc pada API getUserMedia() terjadi *error code*.

- Browser Safari v5.1.7

"Your browser doesn't appear to support WebRTC". Hal ini menunjukkan bahwa browser Safari v5.1.7 masih belum support dengan arsitektur webRTC, tetapi browser safari v5.1.7 mampu untuk mengakses website pada server dengan port 8181, dapat diartikan bahwa HTML5 yang digunakan browser safari v5.1.7 masih belum support dengan HTML5 yang diterapkan oleh webRTC.

- Browser Opera v38.0

"Failed to get access to local media. Error code was PermissionDeniedError" sama seperti error pada browser chrome.

- Browser Internet Explorer v8.0

Pada browser internet explorer v8.0 memberikan notifikasi error "The webpage cannot be displayed". Hal ini menunjukkan bahwa browser internet explorer tidak support untuk diterapkan webRTC. Untuk mengakses website yang ada pada server dengan port 8181 masih belum bisa, dapat diartikan bahwa javascript dari internet explorer v8.0 masih belum support dengan javascript yang diterapkan oleh webRTC.

2. Terdapat empat parameter *quality of service* yang telah diuji pada waktu yang berbeda-beda yaitu 2 menit, 4 menit, 6 menit dan 8 menit parameter *quality of service* yang dimaksud adalah rata-rata *delay*, *jitter*, *throughput* dan *packets loss*.

- Rata-rata *Delay*

Dapat disimpulkan bahwa nilai *delay* dari pengujian 2 menit, 4 menit, 6 menit dan 8 menit mempunyai nilai yang konstan yaitu antara 9.970 ms sampai 9.985 dan mempunyai rata-rata *delay* 9.977 ms. Dapat disimpulkan bahwa dengan bandwidth 128 kbps waktu delay pengiriman paket data webRTC audio call yang menggunakan

framework easyRTC dan node.js sebagai media signalingnya serta *codec* opus sebagai transmisi paket data audio memerlukan waktu rata-rata 9.977 ms agar data sampai ketujuan.

- *Jitter*

Nilai jitter yang diperoleh dari pengujian pada rentan waktu 2 menit, 4 menit, 6 menit dan 8 menit adalah berkisar antara 0.000270 ms sampai 0.000395 ms dan mempunyai nilai rata-rata 0.000314 ms. Faktor yang mempengaruhi nilai jitter disebabkan karena terdapatnya variasi-variasi dalam panjang antrian dan lama waktu pengolahan paket data. Jitter disebut juga dengan variasi delay karena pada *latency* terdapat variasi delay pada transmisi data dalam jaringan.

- *Throughput*

Nilai throughput yang dihasilkan dari pengujian pada rentan waktu 2 menit, 4 menit, 6 menit dan 8 menit mempunyai nilai antara 13.462 kbps sampai 14.221 kbps dan mempunyai nilai rata-rata 14.019. perlu diketahui pada pengujian ini menggunakan bandwidth sebesar 128 kbps namun throughput yang dihasilkan hanya 14.019 kbps. Faktor seperti redaman dan kapasitas bandwidth dapat mempengaruhi hasil dari pengukuran ini.

- *Packets Loss*

Nilai packets loss yang diperoleh dari pengujian ini pada rentan waktu 2 menit, 4 menit, 6 menit dan 8 menit menghasilkan nilai persentase antara 2.857044% sampai 5.957055% dan mempunyai nilai rata-rata persentase 4.255159%. faktor yang menyebabkan terjadinya packets loss karena adanya tumbukan atau tabrakan antara data pada jaringan. Dapat juga buffer pada suatu perangkat jaringan kelebihan beban sehingga tidak dapat menampung data baru dan menyebabkan kehilangan data.

Dari nilai-nilai yang telah dihasilkan dalam pengujian ini dengan parameter *delay*, *jitter*, *throughput* dan *packets loss* bahwa webRTC yang menerapkan framework easyRTC dan node.js sebagai media *signaling* serta *codec* opus sebagai *codec* untuk transmisi suara dan juga menggunakan *bandwidth* sebesar 128 kbps, dapat disimpulkan bahwa sudah cukup baik untuk melakukan komunikasi VOIP dengan metode webRTC, karena mempunyai nilai indeks *delay* sangat baik dan nilai indeks *jitter* yang baik menurut versi TIPHON, namun pada nilai indeks *throughput* yang jelek dan nilai indeks *packets loss* yang sedang menurut versi TIPHON.

3. DAFTAR PUSTAKA

[1] Brett McLaughlin, 6 Juli 2011. "*What is Node.js?*", publisher O'Reilly Media. http://radar.oreilly.com/2011/07/what-is-node.html.

[2] Muhammad Iqbal C. R., Muchammad Husni dan Hudan Studiawan. Sept, 2012. "*Implementasi Klien SIP Berbasis Web Menggunakan HTML5 dan Node.js*". http://ejurnal.its.ac.id/index.php/teknik/article/view/643.

[3] Phil Edholm, E. Brent Kelly, Ph.D., Matt Krebs, November 2014, "*WebRTC Ecosystem "Overview" A Description of the Ecosystem Framework*", pedholm@pkeconsulting.com, bkelly@kelcor.com, mattkrebs@kelcor.com.

[4] Rob Manson, September 2013, "*Getting Started with WebRTC*", www.packtpub.com.

[5] Erick Linask, May 2013, "*Making the Right Signalling Choice*", http://docslide.us/technology/webrtc-conference-and-expo-may-2013-making-the-right-signalling-choice.html.

[6] Dewi Yolanda S. A., Dr. Ir. Sholeh Hadi Pramono, MS., M. Fauzan Edy Purnomo, ST., MT., 2013, "*Simulasi Kinerja Routing Protokol Open Shortest Path First (OSPF) dan Enhanced Interior Gateway Routing Protocol (EIGRP) Menggunakan Simulator Jaringan Opnet Modeler v. 14.5*" http://elektro.studentjournal.ub.ac.id/index.php/teub/article/view/79.

[7] Tiphon. "*Telecommunication and Internet Protocol Harmonization Over Network (TIPHON) General Aspec of Quality of Service (QoS)*", DTR/TIPHON-05006 (cb0010cs.PDF).1999.

[8] Yanto. 2014, "*ANALISIS QOS (QUALITY OF SERVICE) PADA JARINGAN INTERNET (STUDI*

KASUS: FAKULTAS TEKNIK UNIVERSITAS TANJUNGPURA)".

[9] Noviyanti K, St.Rukmini Adikarini, Muhammad Nizwar, Mukarramah Yusuf, 9 Agustus 2014, "*PENGEMBANGAN APLIKASI LIVE CHAT DENGAN MENGGUNAKAN WEBRTC SEBAGAI PEMANFAATAN TEKNOLOGI INFORMASI DAN KOMUNIKASI UNTUK MEDIA PEMBELAJARAN JARAK JAUH (E-LEARNING)*".

[10] Ben Feher, Lior Sidi, Asaf Shabtai, Rami Puzis, 2 Januari 2016, "*The Security of WebRTC*", http://arxiv.org/abs/1601.00184.

[11] Jean-Marc Valin, Gregory Maxwell, Timothy B. Terriberry, and Koen Vos, Oktober 2013, "*High-Quality, Low-Delay Music Coding in the Opus Codec*".

[12] William S. Bubanto, Arie S. M. Lumenta, Xaverius Najoan, 2014, "*Analisis Kualitas Layanan Jaringan Internet (Studi Kasus PT. Kawanua Internetindo Manado)*".